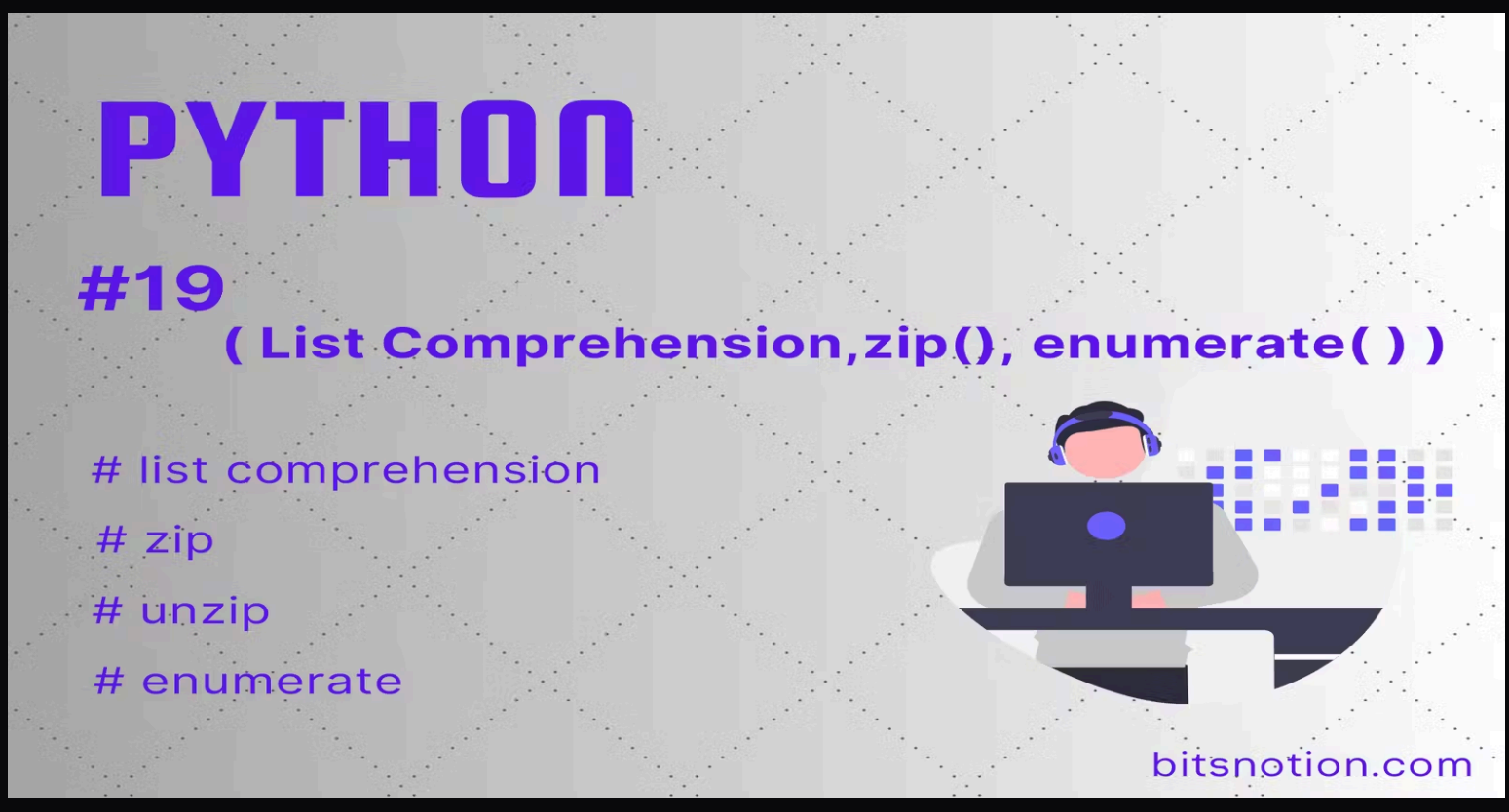

List comprehension, zip(), & enumerate( )

Nirmal Pandey · Apr 3, 2024 · 3 min read

Table of contents

List comprehension

› Some examples of List Comprehensions :

Zip( )

Unzipping

Enumerate( )

List comprehension

One of the most useful tools in Python is list comprehension. It is a concise and efficient way to create a new list based on the values in an existing iterable object. List comprehensions take the following form:

```
my_list = [expression for element in iterable if condition]
```

In this syntax:

- **expression**: What you want to do with each element. This can be a transformation or any operation.
- **element**: The variable name that represents each item in the iterable.
- **iterable**: The sequence (like a list, tuple, string, etc.) you are iterating over.
- **condition** (optional): Any expression that evaluates to `True` of `False`.

Some examples of List Comprehensions :

1. Creating a List of Squares:

COPY

```
numbers = [1, 2, 3, 4, 5]
squared = [x ** 2 for x in numbers]
print(squared)  # Output: [1, 4, 9, 16, 25]
```

2. Filtering Even Numbers:

COPY

```
numbers = [1, 2, 3, 4, 5]
even = [x for x in numbers if x % 2 == 0]
print(even)  # Output: [2, 4]
```

3. Manipulating Elements with Conditions:

COPY

```
names = ["Alice", "Bob", "Charlie"]
formatted_names = [name.upper() if len(name) > 4 else name.lower() for name in na
print(formatted_names)  # Output: ['alice', 'BOB', 'CHARLIE']
```

4. Flattening a Nested List:

COPY

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened = [x for sublist in nested_list for x in sublist]
print(flattened)  # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Zip( )

The `zip()` function is used to combine two or more iterables (like lists or tuples) into a single iterable of tuples. Each tuple contains elements from the input iterables that have the same position.

```python
names = ['Alice', 'Bob', 'Charlie']
scores = [85, 92, 78]

combined = zip(names, scores)
print(list(combined))  # Output: [('Alice', 85), ('Bob', 92), ('Charlie', 78)]
```

In this example, `zip()` pairs each element from `names` with the corresponding element from `scores`, creating a list of tuples. This can be particularly useful for iterating over multiple lists simultaneously.

Unzipping

You can also unzip an object with the `*` operator. Here's the syntax:

```python
cities = [('New York', 8419000), ('Los Angeles', 3980400), ('Chicago', 2716000)]
city_names, populations = zip(*cities)
print(city_names)     # Output: ('New York', 'Los Angeles', 'Chicago')
print(populations)    # Output: (8419000, 3980400, 2716000)
```

Enumerate( )

The `enumerate()` function adds a counter to an iterable and returns it as an enumerate object. This is extremely useful when you need both the index and the value of elements in a loop.

```python
fruits = ['apple', 'banana', 'cherry']

for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")

#Output :
'''
0: apple
1: banana
2: cherry
'''
```

With `enumerate()`, you no longer need to manage a separate counter variable, making your loop cleaner and more readable.